REPAST MODEL TESTING GUIDE

JONATHAN OZIK, NICK COLLIER - REPAST DEVELOPMENT TEAM

0. Before we Get Started

Before we can do anything with Repast Simphony, we need to make sure that we have a proper installation of Repast Simphony 2.1. Instructions on downloading and installing Repast Simphony on various platforms can be found on the Repast website.

1. Getting Started with Repast Simphony Model Testing

This guide will walk you through a few model testing use cases using Repast Simphony. To learn more about model testing, including the benefits of test driven development (TDD) when developing agent models, see Collier and Ozik (2013)[1]. For more information on the JUnit testing framework that we will be using, see http://junit.org.

To add tests into an existing Repast Simphony project, we recommend the following setup steps:

(1) Add a `test` source folder to the project. This can be accomplished in a number of ways. One way is to right click on the project and navigate to *New* → *Source Folder* (Fig. 1). Then fill in `test` in the *Folder name* text field and click on *Finish* (Fig. 2).

(2) Modify the output folder for the `test` source folder. Right click on the project and navigate to *Properties*. Then choose the *Java Build Path* entry in the left bar and select the *Source* tab. Select the check box that reads *Allow output folders for source folders* and expand the `test` entry (Fig. 3). Select the *Output folder* entry and click on the *Edit* button (Fig. 4). Then select the *Specific output folder* option and input `testbin`, or something else different from `bin` (Fig. 5). Click on *Okay* and then *Okay* again.

(3) Add a JUnit Test Case by right clicking on the project in the Package Explorer view, choosing *New* → *Other*... (Fig. 6). Then navigate to *Java* → *JUnit* and choose *Junit Test Case* (Fig. 7)[2]. In the *New JUnit Test Case* wizard choose the *New JUnit 4 test* option, specify the correct source folder (`test`), name your test case, include all the method stubs, and click on *Finish* (Fig. 8)[3]. If JUnit was not

Date: June 22, 2013.

[1]Collier, N, and J Ozik. Test-Driven Agent-Based Simulation Development. To appear in WSC 2013 Proceedings. Washington, D.C., 2013.

[2]This creates a Java JUnit test case, but a Groovy JUnit test case can be used as well.

[3]While this document is focused on JUnit 4, JUnit 3 or other testing frameworks can also be utilized.

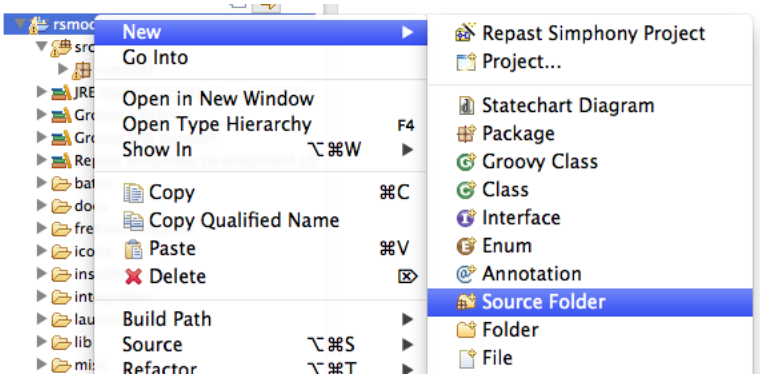

FIGURE 1. Selecting your project, right-clicking and choosing *New* → *Source Folder*.

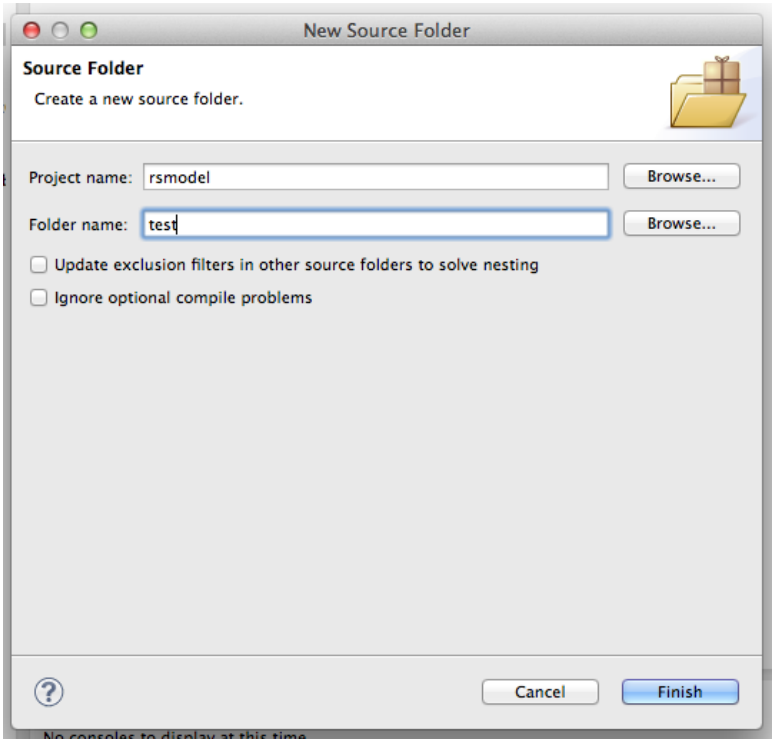

FIGURE 2. New source folder wizard.

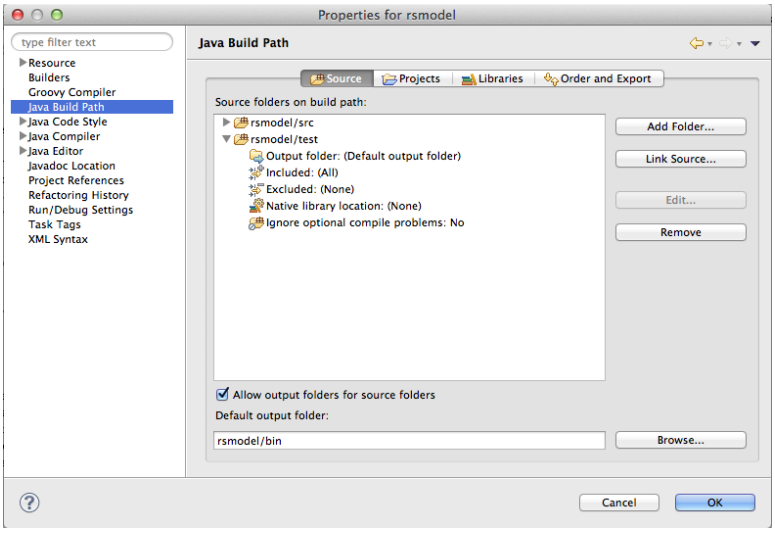

FIGURE 3. *Java Build Path* → *Source* tab in a project's properties.

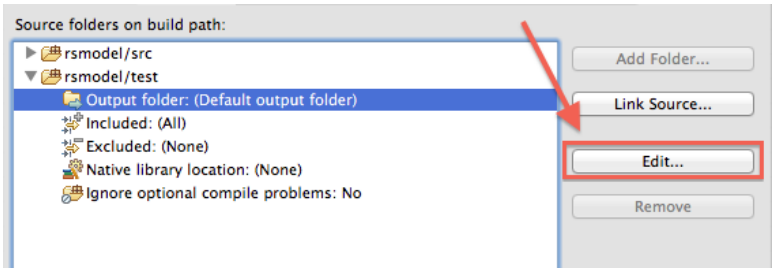

FIGURE 4. Edit the output folder of the `test` source folder.

previously on your project's build path, you will see a dialog asking if you'd like to add it (Fig. 9). Click *OK* to add the JUnit 4 library to the project's build path.

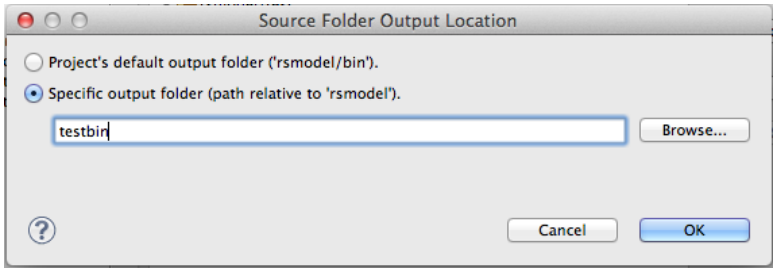

FIGURE 5. Choosing the source folder output location.

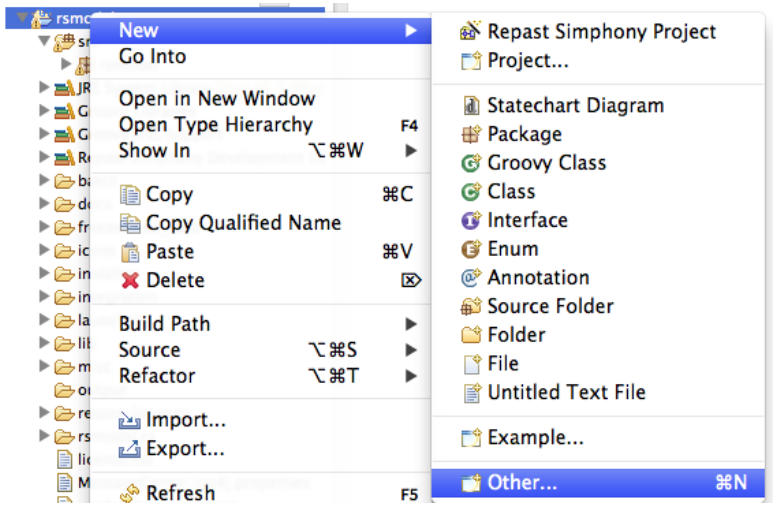

FIGURE 6. Selecting your project, right-clicking and choosing New → Other... .

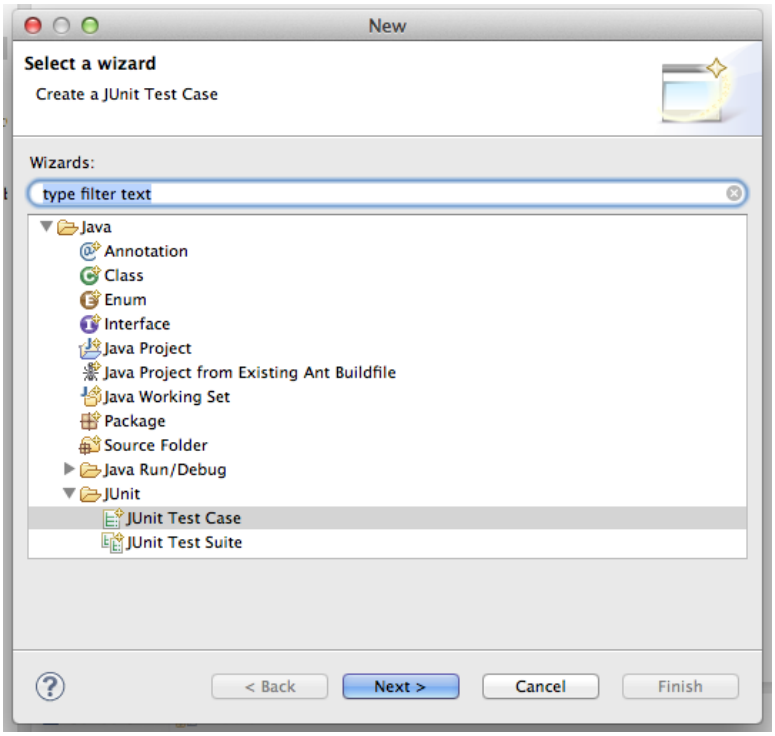

FIGURE 7. The JUnit Test Case option within Java → JUnit.

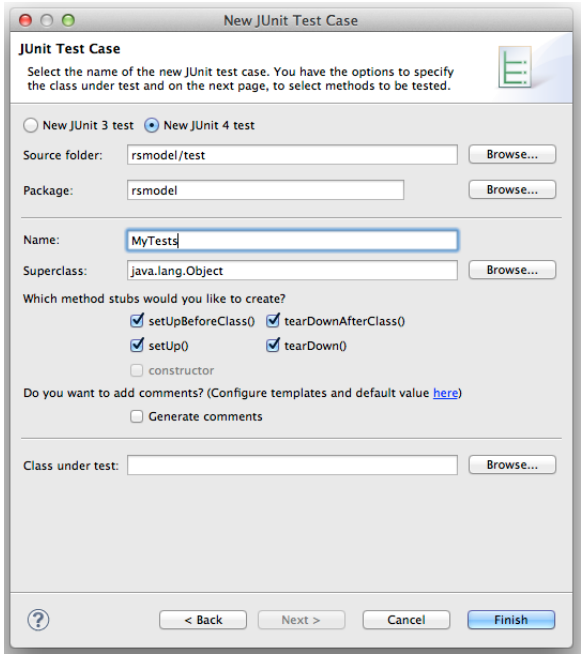

FIGURE 8. The new *JUnit Test Case* wizard.

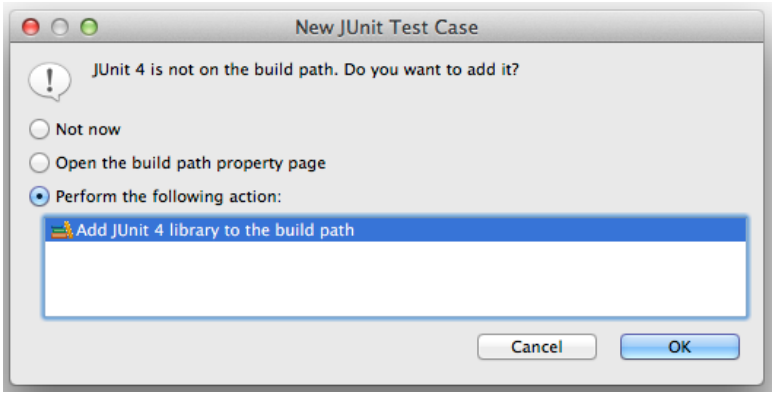

FIGURE 9. Add JUnit 4 library to build path.

Once the setup is complete, there are a number of different types of model tests that can be created, based on the nature of the model behavior that is being tested. We will go over a few such cases next.

1.1. **Use Case 1: Simple Unit Testing with Repast Simphony Models.** If the elements being tested are relatively decoupled, there is nothing special that needs to be done in terms of test case setup. In this scenario, the @BeforeClass, @AfterClass, @Before, and @After annotated methods do not need any Repast Simphony specific elements and tests can be written in the usual way JUnit tests are written.

1.2. **Use Case 2: Schedule Based Model Testing with Repast Simphony Models.** Some model behaviors that you might want to test involve scheduled behaviors. For example, you might want to know that at a certain tick, specific model actions had occurred. This is especially relevant to any Statecharts related behaviors. In this situation, the RunEnvironment will need to be passed a Schedule object. Listing 1 shows an example where the schedule based setup is executed before each of the tests that will be run (i.e., in the @Before annotated method).

```
1 @Before
2 public void setUp() throws Exception {
3    Schedule schedule = new Schedule();
4    RunEnvironment.init(schedule, null, null, true);
5    Context context = new DefaultContext();
6    RunState.init().setMasterContext(context);
7
8    // Any additional setup
9 }
```

LISTING 1. @Before setup method in a schedule dependent test case.

Line 3 in Listing 1 shows the creation of the Schedule object. It is then sent as the first parameter to the RunEnvironment static `init` method. The second and third parameters can be `null` for this case and the fourth parameter indicates whether this is a batch (i.e., headless) run, which it is so we specify `true`. Line 5 creates a new DefaultContext that is set as the master context of the `init`-ed RunState in line 6. Lines 5 and 6 are only strictly necessary for testing behaviors that rely on a master context being set, which is the case for Statecharts.

At this point, tests can be written such as in Listing 2:

```
@Test
public void testUninfectedToInfected() {
   ISchedule schedule = RunEnvironment.getInstance().
         getCurrentSchedule();
   Person p = new Person();
   assertEquals(UNINFECTED, p.getStatus());
   for (int i = 0; i < 5; ++i) {
      schedule.execute();
   }
   assertEquals(INFECTED, p.getStatus());
}
```

LISTING 2. Example test method where the schedule is advanced.

In this hypothetical example, a Person agent is created, it is verified that the person's status is `UNINFECTED`, the schedule is advanced 5 times, at which point the person's status is checked to see that it is `INFECTED`. An important item to note here is that the scheduler in Repast Simphony doesn't just allow for discrete time steps so the fact that the schedule is executed 5 times doesn't necessarily mean that we will find ourselves at tick 5 after the `for` loop, unless there were actions scheduled only to occur on every tick.

1.3. **Use Case 3: Context Builder Based Model Testing with Repast Simphony Models.** For cases where the specific setup defined in a ContextBuilder is required for testing, the @Before testing setup might look like Listing 3.

```
. . .
public Context context;

@Before
public void setUp() throws Exception {
   context = new DefaultContext();
   MyContextBuilder builder = new MyContextBuilder();
   context = builder.build(context);

   // Any additional setup
}
. . .
```

LISTING 3. @Before setup method in a context builder dependent test case.

Here, after creating the DefaultContext, the context builder is used to build it into the state it should be in at the start of each simulation run. A test utilizing the setup in Listing 3 might look like Listing 4. In this hypothetical test, a new Person agent is added

to the main context and a `hasNeighbors` method is called to ensure that the added Person agent has neighbors in the pre-built context.

```
1 @Test
2 public void testAddingPersonToContext() {
3    Person p = new Person();
4    context.add(p);
5    assertTrue(p.hasNeighbors());
6 }
```

LISTING 4. Example test method where the context state created by a context builder is used.

1.4. **Use Case 4: Model Testing with ReLogo Models.** ReLogo models come with the infrastucture and associated assumptions of the ReLogo world which is built by the SimBuilder context builder included with each model. To test a ReLogo model there are a few additional steps needed for the test setup beyond what was done in Section 1.3. Listing 5 shows the setup (optionally) separated into @BeforeClass and @Before components. The idea here is that the ReLogo world is built once before all the tests are run but before each individual test is executed, the observer clears the ReLogo world state, removing existing turtles and links and resetting patches.

```
1  . . .
2  static UserObserver observer;
3
4  @BeforeClass
5  public static void setUpBeforeClass() throws Exception {
6     String scenarioDirString = "ModelName.rs";
7     ScenarioUtils.setScenarioDir(new File(scenarioDirString));
8     File paramsFile = new File(ScenarioUtils.getScenarioDir(),
9        "parameters.xml");
10    ParametersParser pp = new ParametersParser(paramsFile);
11    Parameters params = pp.getParameters();
12    RunEnvironment.init(new Schedule(), null, params, true);
13    Context context = new DefaultContext();
14    SimBuilder builder = new SimBuilder();
15    context = builder.build(context);
16
17    // If statecharts are used in the ReLogo model
18    RunState.init().setMasterContext(context);
19
20    observer = (UserObserver)context.iterator().next();
21
22    // Any additional before class setup
23 }
24
25 @Before
26 public void setUp() throws Exception {
27    observer.clearAll();
28
29    // Any additional setup
30 }
31 . . .
```

LISTING 5. @Setup method in a schedule dependent test case.

Some of the contents of Listing 5 will already look familiar from the previous sections but there are additions that we expand on here:

(1) The scenario directory (the .rs folder) is specified in Line 6. Replace "ModelName.rs" with the name of the scenario directory in your project.
(2) The parameters.xml file within the scenario directory is parsed and passed to the RunEnvironment `init` method. This can be done in non-ReLogo models as well.
(3) If statecharts are used in the ReLogo model, Line 18 should be included as well.